J Sign Process Syst (2008) 53:73–88
DOI 10.1007/s11265-008-0221-7

Configurable LDPC Decoder Architectures for Regular and Irregular Codes

Marjan Karkooti · Predrag Radosavljevic · Joseph R. Cavallaro

Received: 12 January 2007 / Revised: 5 September 2007 / Accepted: 21 April 2008 / Published online: 10 May 2008

Abstract Low Density Parity Check (LDPC) codes are one of the best error correcting codes that enable the future generations of wireless devices to achieve higher data rates with excellent quality of service. This paper presents two novel flexible decoder architectures. The first one supports (3, 6) regular codes of rate 1/2 that can be used for different block lengths. The second decoder is more general and supports both regular and irregular LDPC codes with twelve combinations of code lengths −648, 1296, 1944-bits and code rates- 1/2, 2/3, 3/4, 5/6- based on the IEEE 802.11*n* standard. All codes correspond to a block-structured parity check matrix, in which the sub-blocks are either a shifted identity matrix or a zero matrix. Prototype architectures for both LDPC decoders have been implemented and tested on a Xilinx field programmable gate array.

Keywords Low density parity check codes · Reconfigurable architectures · Error correcting codes

1 Introduction

Low Density Parity Check (LDPC) codes were proposed by Gallager [1] more than 40 years ago, but his work received little attention until after the invention of turbo codes, which used the same concept of iterative decoding. In 1996, MacKay and Neal [2] rediscovered LDPC codes. The excellent error correction capability of these codes led to an overwhelming interest in designing LDPC codes suitable for practical applications.

One of the major applications for error correcting codes such as convolutional code, turbo or LDPC codes is wireless communications. Every year millions of wireless devices enter the market. There is an increasing need for higher data rate and higher throughput which requires the use of stronger and more powerful error correcting codes. The LDPC and turbo codes have been shown to be the two major competitors in this space. Many standards have adopted or are considering LDPC codes, such as DVB-S2, IEEE 802.11*n* and WiMAX. Other standards, such as DVB-S and 3GPP-LTE, use turbo codes.

Throughput and complexity of LDPC decoding depend on six major parameters: the number of bits in the codeword or 'block length', the ratio of the number of information bits to the codeword length or 'code rate', the complexity of computations at each processing node, the complexity of interconnection, the parallelism level, and the number of times that local computations need to be repeated or the number of iterations. There is a trade-off between performance of the decoder and the complexity and speed of decoding. We will address these trade-offs throughout this paper in more detail. One of the main advantages of LDPC codes is the inherent parallelism in the decoding process which can lead to a very high decoding throughput if used properly. In this paper, we present two novel high throughput LDPC decoders. One of them supports the (3, 6) regular LDPC codes of rate 1/2 [3]. This

M. Karkooti (✉) · P. Radosavljevic · J. R. Cavallaro
Department of Electrical and Computer Engineering,
Rice University, 77005 Houston, TX, USA
e-mail: marjan@rice.edu

P. Radosavljevic
e-mail: rpredrag@rice.edu

J. R. Cavallaro
e-mail: cavallar@rice.edu

architecture is suitable for applications with area/power constraints and high throughput requirements which do not need total flexibility in terms of block length and code rate. By using the knowledge gained from the first decoder architecture, we describe the design of a high throughput, flexible LDPC decoder that supports multiple block lengths (648, 1296, 1944 bits) and multiple code rates (1/2, 2/3, 3/4, 5/6) based on the IEEE 802.11*n* standard [4] for irregular LDPC codes. These codes are block-structured with profiles (refer to [5]) that provide error-correcting performance close to excellent fully random codes, such as the codes from [6]. This LDPC decoder has a semi-parallel design and supports twelve codes with a small control and arithmetic logic overhead [7]. It can also switch between different code sizes and code rates on the fly without any need for recompilation or re-synthesis. Furthermore, it uses layered belief propagation (LBP) which converges twice as fast as the standard belief propagation algorithm, resulting in two times higher data throughput [8]. This decoder architecture is easily expandable to support other code sizes/rates.

In the last few years extensive research have been done on designing architectures for LDPC coding. Researchers have been looking for the best trade-off between area, data rate and power consumption. Authors in [9] directly mapped the Sum-Product decoding algorithm to hardware. They used a fully parallel approach and connected all the functional units with wires according to the Tanner graph connections. Although this decoder has very good performance, the large area and the routing complexity and overhead makes this approach infeasible for larger block lengths (e.g. more than 1000 bits). Also, there is no flexibility in this decoder.

Another approach is to have a semi-parallel decoder and reuse the processing units to decrease the area cost at the expense of lower throughput than fully parallel. In [10], a field programmable gate array (FPGA) implementation of a (3, 6) regular LDPC semi-parallel decoder is offered. They used a multi-layered interconnection network to access messages from memory. Authors in [11] proposed a low-power 1055 bit, rate 0.4, (3, 5) regular semi-parallel decoder architecture. A fully structured parity check matrix (PCM) was used to eliminate the data dependence among the neighboring nodes which led to a simpler memory addressing scheme than [10]. By using a semi-parallel approach, several processing units - equal to the size of the sub-blocks *S*- can work simultaneously to reduce the decoding time and hence increase the throughput by a factor of *S* compared to the serial approach.

Most of the works related to architectures for LDPC decoders are based on a fixed code [3, 12] or they are scalable and support multiple rates, but only for a single code size. For example, a scalable structured LDPC decoder with relatively high data throughput is proposed in [13] for very long codeword lengths defined by the DVB-S2 standard. Also, authors in [14, 15] and [16] offered multi-rate decoder designs based on structured PCMs for regular and irregular codes, using the standard belief propagation algorithm.

In recent years, several other architectures have been proposed for decoding LDPC codes. Most of them use Quasi-cyclic LDPC codes with variations of the sum-product decoding algorithm. To name a few, the reader is referred to [17–26].

FPGAs or Application Specific Integrated Circuits (ASICs) are very suitable for designing LDPC encoders/decoders because of the flexible number of processing units that can work in parallel rather than the limited number of processing units in Digital Signal Processors or general purpose processors.

Every application has its own requirements and specific parameters. In order to decrease the design time and reuse the same hardware for many applications, a general, configurable, flexible architecture that supports several cases is of interest. This LDPC decoder should support a family of codes and should be able to switch between different code types seamlessly. For example, in wireless communication systems, or in user cooperation schemes, when the transmission channel is good, codes with a fewer number of redundant bits or higher rates can be used. Or, smaller block lengths can be used when there is less data to be transferred. So, a multi-size, multi-rate LDPC decoder is a suitable choice for future generations of wireless devices.

The organization of the paper is as follows: Section 2 overviews the LDPC codes, describing their encoding and decoding processes. A flexible architecture for regular LDPC codes is described in Section 3. Section 4 presents a novel flexible architecture for both regular and irregular LDPC codes. The decoder architecture prototypes are implemented on an FPGA and also synthesized for ASIC using Chartered Semiconductor 0.13 μm CMOS technology. The details of the architectures are presented in Sections 3 and 4. Section 5 concludes the paper.

2 LDPC Codes

LDPC codes are a class of linear block codes specified by a very sparse PCM $H_{(N-K)\times N}$. Each element of the

PCM is either a *zero* or a *one*, where nonzero entries are typically placed at random. During the encoding process, $N-K$ redundant bits are added to the K information bits to form a codeword length of N bits. The code rate is the ratio of the information bits to the total bits transmitted in a codeword ($R = K/N$). LDPC codes are usually represented by a bi-partite graph called a 'Tanner graph'. There are two classes of nodes in a Tanner graph, 'Variable' nodes and 'Check' nodes. A variable node corresponds to a 'coded bit' or a PCM column, and a check node corresponds to a parity check equation or a row of the PCM. There is an edge between each pair of nodes if there is a 'one' in the corresponding PCM entry. During the decoding process, messages are passed among the graph nodes. Log-likelihood ratios (LLRs) are used for representation of reliability messages.

The number of nonzero elements in each row or column of a PCM is called the 'degree' of that node. An LDPC code is regular or irregular based on the node degrees. If variable or check nodes have different degrees, then the LDPC code is called 'irregular' otherwise, it is called 'regular'. In low SNR regimes, irregular codes usually have better performance than regular codes [5] because the nodes with higher degrees converge faster and assist the nodes with lower degrees. On the other hand, irregularity of the code results in a more complex hardware architecture.

Figure 1 shows a Tanner graph of a simple PCM $H_{4\times 8}$. In this graph variable and check nodes have degrees of two and four, respectively. To encode a message M of K bits with LDPC codes, one might compute $C_{1\times N} = M_{1\times K}.G_{K\times N}$, in which C is the codeword and G is the generator matrix of the code. At first glance, encoding may seem to be a computationally expensive task for large matrices, but there exist some reduced complexity algorithms for encoding the LDPC codes such as [27]. In this work, our focus is on the decoder design. In the following sections, we will show that by taking advantage of the iterative decoding process and structure of the code matrix, we can increase the parallelism level of the architecture and increase the throughput of the decoder.

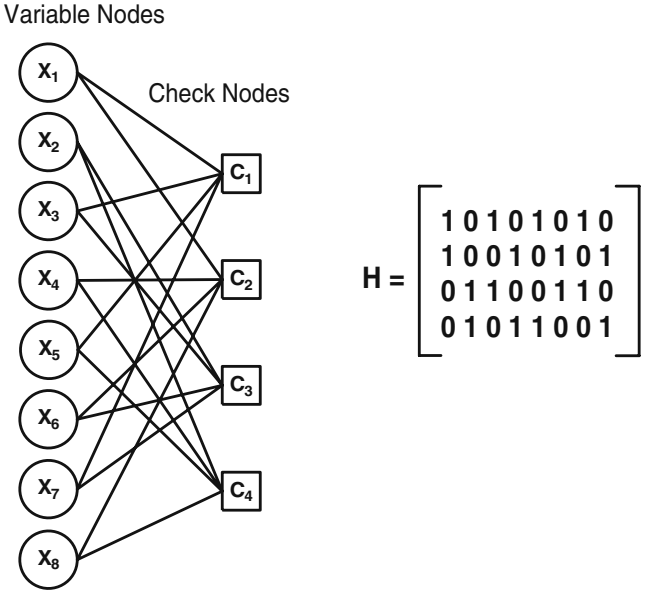

Figure 1 Tanner graph of a PCM.

2.1 Decoding Using Variations of Sum-Product Algorithm

Gallager presented a decoding algorithm that is effectively optimal in his seminal work in 1960. Since then, other researchers have independently discovered that algorithm and related algorithms, albeit sometimes for different applications. The algorithm is usually referred to with different names such as 'Message Passing', 'Sum-Product' or 'Belief Propagation'. It iteratively computes the probability distributions of variables in graph-based models. The iterative decoding algorithm for turbo codes is a specific instance of the Sum-Product algorithm.

Consider the Tanner graph of Fig. 1 for an LDPC code. During the decoding process, information is sent along the edges of the graph. Local computations are done in each node of the graph. To facilitate the subsequent iterative processing, one tries to keep the graph as sparse (low density) as possible. Although that approach can be suboptimal, it is usually quite close to optimal and has an excellent complexity vs. performance tradeoff. In the rest of this section, we will describe the original Sum-Product algorithm and some of its variations such as the Modified Min-Sum (MMS) algorithm and LBP algorithm. Throughout this paper, an Additive White Gaussian Noise channel is considered with Binary Phase Shift Keying modulation of the signals.

2.1.1 Sum-Product Algorithm in Log Domain

Let R_{mj} denote the check node LLR message sent from the check node m to the variable node j. Let $L(q_{mj})$ denote the variable node LLR message sent from the variable node j to the check node m. The $L(q_j)$ ($j = 1, \ldots, N$) represent the *a posteriori* probability ratio (APP messages) for all variable nodes (coded bits). The APP messages are initialized with the corresponding *a*

priori (channel) reliability value of the coded bit j. In each iteration, some of the APP messages corresponding to the code bits that are received in error are corrected. Theoretically, the received codeword converges to the transmitted codeword after several iterations.

The Sum-Product algorithm operates on the nonzero entries of the PCM H. It iterates over the columns and rows of the PCM performing the following steps:

Step 1) Initialization: For each variable node j, messages $L(q_{mj})$ that correspond to a particular check equation m are computed according to:

$$L(q_{mj}) = L(q_j) = 2y_j/\sigma^2, \tag{1}$$

in which y_j is the received signal and σ is the channel noise.

Step 2) For each check node m, messages R_{mj}, corresponding to all variable nodes j that participate in a particular parity-check equation, are computed according to:

$$R_{mj} = \prod_{j' \in N(m)\setminus\{j\}} \operatorname{sign}\left(L(q_{mj'})\right) \times \phi\left[\sum_{j' \in N(m)\setminus\{j\}} \phi\left(L(q_{mj'})\right)\right], \tag{2}$$

where $N(m)$ is the set of all variable nodes from parity-check equation m and $\phi(x)$ is equal to:

$$\phi(x) = -\log\left(\tanh(x/2)\right) = \log\left(\frac{e^x+1}{e^x-1}\right). \tag{3}$$

Equation 2 is performing summations and multiplications of different messages and hence the name sum-product.

Step 3) Compute messages at variable nodes and pass to check nodes,

$$L(q_{mj}) = L(q_j) + \Sigma_{m' \in C(j)\setminus\{m\}} R_{m'j}. \tag{4}$$

Step 4) The *a posteriori* reliability messages are updated according to:

$$L(Q_j) = L(q_j) + \Sigma_{m \in C(j)} R_{mj}. \tag{5}$$

Step 5) Threshold the values calculated in each variable node to find a codeword. For every row index j:

$$\hat{c}_j = \begin{cases} 1 & if\ L(Q_j) < 0 \\ 0 & else. \end{cases} \tag{6}$$

If the codeword satisfies all the parity check equations or if the maximum number of iteration is reached then stop, otherwise continue iterations from Step 2.

Min-Sum Algorithm The Min-Sum algorithm is an approximation of the sum-product algorithm in which a set of calculations on a nonlinear function $\phi(x)$, is approximated by a minimum function. Consider the update equation for R_{mj} in the Sum-Product algorithm (Eq. 2). The term $\phi(x)$ is a monotonically decreasing function for the values of $x > 0$ (See Fig. 2). It is intuitive that the term corresponding to the smallest $\|L(q_{mj'})\|$ in the above summation in Eq. 2 dominates, so that:

$$\phi\left(\Sigma_{j' \in N(m)\setminus\{j\}} \phi\left(\|L(q_{mj'})\|\right)\right) \approx \phi\left(\phi\left(min_{j'}\|L(q_{mj'})\|\right)\right) = min_{j'}\|L(q_{mj'})\|. \tag{7}$$

The second equality follows from $\phi(\phi(x)) = x$. Comparing the Min-Sum algorithm to the Sum-Product algorithm, Eq. 4 in the Sum-Product algorithm is replaced by the following approximation in Min-Sum:

$$R_{mj} \approx \left(\prod_{j' \in N(m)\setminus\{j\}} sign(L(q_{mj'}) \times min_{j' \in N(m)\setminus\{j\}}\|L(q_{mj'})\|\right). \tag{8}$$

Because of this approximation, there is some degradation in the performance of the Min-Sum algorithm compared to the Sum-Product algorithm. The error correcting performance of the Min-Sum algorithm can be improved by adding an offset to, or by scaling the soft information; the new variation of the algorithm is called MMS algorithm.

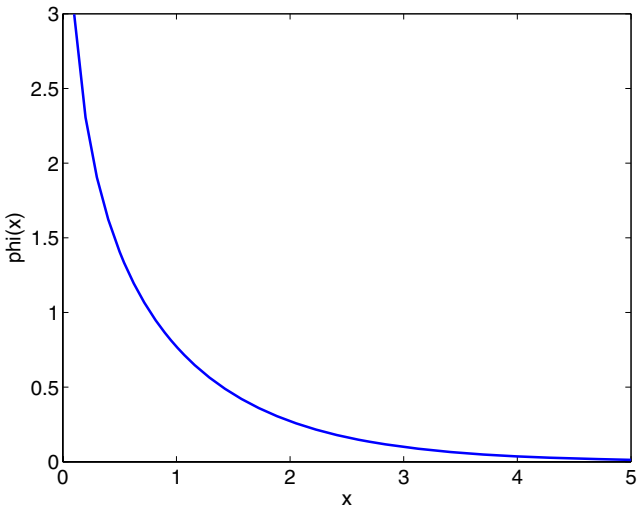

Figure 2 The $\phi(x) = -\log(\tanh(x/2))$ function which is part of the Log-Sum-Product algorithm.

MMS Algorithm In the literature, it has been experimentally shown that scaling the soft information during the decoding step using the Min-Sum algorithm results in better performance. Scaling slows down the convergence of iterative decoding and reduces the overestimation error compared with the Sum-Product algorithm. Heo [28] showed that density evolution techniques can be used to determine the optimal scaling factor. He also showed that for a $(3, 6)$ LDPC code a scaling factor of 0.8 is optimal. In this algorithm, it is enough to change Step 2 in the Min-Sum algorithm to:

$$L(q_{mj}) = \left(L(c_j) + \Sigma_{m' \in C(j)\setminus\{m\}} L(r_{m'j})\right) * \gamma, \tag{9}$$

in which γ is the scaling factor. Also, some researchers have shown that adding a correcting factor to the Min-Sum equation has the same effect as scaling [29].

Figure 3 shows a comparison between the performance of Sum-Product, Min-Sum and MMS algorithms. It can be seen that scaling the soft information not only compensates for the loss of performance because of approximation, but also results in superior performance compared to the Sum-Product algorithm, because of the reduction in overestimation error. MMS with scaling is used as the decoding algorithm in our first architecture and Layered belief propagation with MMS with correcting offset (LBP-MMS) is used in the second architecture. We will describe the LBP-MMS algorithm in more detail in the next section.

Table 1 shows a comparison between the number of calculations needed for each of the decoding algorithms for a $(3, 6)$ LDPC code in each iteration of decoding. It can be observed that the MMS algorithm substitutes the costly function evaluations with addition and shift operations. Although the MMS algorithm has a few more additions than other algorithms, it is still preferred since nonlinear function evaluations are omitted. The function $\phi(x)$ is sensitive to quantization error which results in loss of decoder performance. Either direct implementation or look up tables can be used to implement this function. Direct implementation is costly for hardware [9]. Look-up tables (LUT) are very sensitive to the number of quantization bits and number of LUT values [10]. Since in each functional unit several LUTs should be used in parallel, they can take a large area on the chip. Eliminating the need for this function in the decoder saves some area and complexity.

Table 1 Complexity of different algorithms per iteration for a (3,6) regular LDPC code.

Algorithm	Addition	$\phi(x)$	Shift
Log-Sum-Prod.	$24(N-K)+7N$	$12(N-K)$	–
Min-Sum	$24(N-K)+7N$	–	–
Mod.Min-Sum	$24(N-K)+10N$	–	$6N$
Layered BP with Mod.Min-Sum	$28(N-K)$	–	–

BP belief propagation

2.1.2 Layered Belief Propagation Algorithm

One of the newer versions of LDPC decoder architectures is based on the LBP algorithm as defined in [11]. This algorithm converges twice as fast as the original Sum-Product algorithm due to the optimized scheduling of reliability messages [30]. The PCM can be viewed as a group of concatenated horizontal layers as shown in Fig. 4, where every layer represents the component code. The belief propagation algorithm is repeated for each horizontal 'Layer' and updated *a posteriori* probabilities are passed between layers. In this

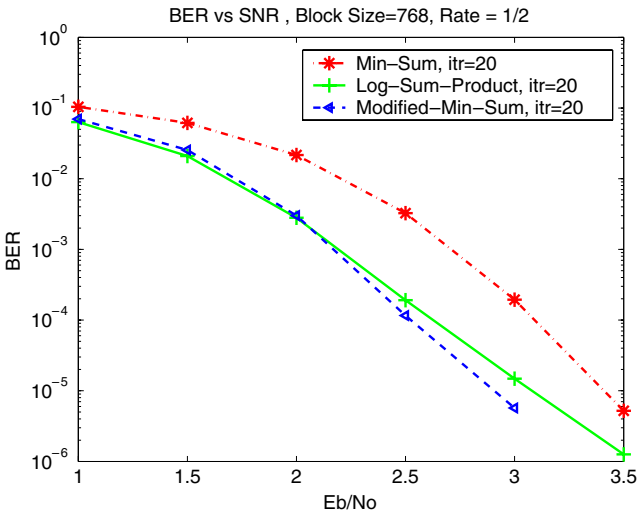

Figure 3 Comparison of different decoding algorithms.

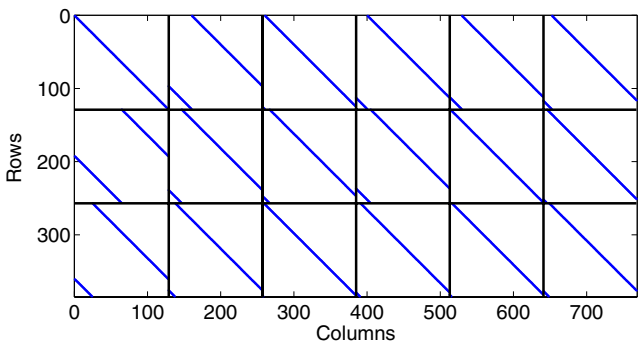

Figure 4 PCM of a regular (3,6) LDPC code.

algorithm, the Eqs. 1, 2, 4, 5 are performed for each layer separately. They are replaced with:

$$L(q_{mj}) = L(q_j) - R_{mj}, \tag{10}$$

$$R_{mj} = \prod_{j' \in N(m)\backslash\{j\}} \operatorname{sign}\left(L(q_{mj'})\right) \phi\left[\sum_{j' \in N(m)\backslash\{j\}} \phi\left(L(q_{mj'})\right)\right], \tag{11}$$

$$L(q_j) = L(q_{mj}) + R_{mj}. \tag{12}$$

The MMS algorithm with correcting offset can be performed for LBP also [29]. In this case, the updating of check node messages in the mth row of the kth decoding iteration is determined as:

$$R_{mj} \approx \prod_{j' \in N(m)\backslash\{j\}} \operatorname{sign}\left(L(q_{mj'})\right) \times \max\left(\min_{j' \in N(m)\backslash\{j\}} |L(q_{mj'})| - \beta, 0\right), \tag{13}$$

where β is a correcting offset equal to a positive constant. Hard decisions can be made after every horizontal layer based on the sign of $L(q_j)$, $j = 1, \ldots, n$. If all parity-check equations are satisfied or the predetermined maximum number of iterations is reached, then the decoding algorithm stops. Otherwise, the algorithm repeats from Eq. 10 for the next horizontal layer.

3 Architecture Design for a Regular LDPC Decoder

Recent hardware designs of semi-parallel LDPC decoders are mostly based on block-structured PCMs, for both regular and irregular codes. The block structure has several advantages over randomly generated codes. First, it simplifies the interconnection network between the variable and check nodes and allows using permuters for address interleaving. Second, instead of having random addressing, addresses can be generated using linear equations. Third, it allows packed storage of several messages in a single memory word. Authors in [5] showed that $(3, 6)$ is the best choice for a rate $1/2$ LDPC code. We have used a $(3, 6)$ code in our first design and the decoding algorithm is the MMS algorithm in the log domain.

In each iteration of the decoding, first all the check nodes receive and update their messages and then, in the next half-iteration all the variable nodes update their messages. If we choose to have a one-to-one relation between processing units in the hardware and variable and check nodes in the Tanner graph, then the design will be fully parallel. Obviously, a fully parallel approach takes a large area; but is very fast. There is also no need for central memory blocks to store the messages. They can be latched close to the processing units [9]. With this approach, the hardware design can be fixed to relate to a special case of the PCM.

Implementing the LDPC decoding algorithm in a fully-serial architecture has the smallest area since it is sufficient to have just one Variable Functional Unit (VFU) and one Check Functional Unit (CFU). The fully-serial approach is suitable for Digital Signal Processors in which there are only a few functional units available to use. However, speed of the decoding is very low.

To balance the trade-off between area and throughput, the best strategy is to have a semi-parallel design. This involves the creation of "l_c" CFUs and "l_v" VFUs, in which $l_c << N - K$ and $l_v << N$ and reuse these units throughout the decoding process. The PCM should be structured in order to enable re-usability of units. Also, in order to design a fast architecture for LDPC decoding, we should first design a good PCM matrix which results in good performance with a small number of decoding iterations. Following the block-structured design, we have designed H matrices for $(3, 6)$ LDPC codes.

Figure 4 shows the regular structured PCM that has been used in this paper. The matrix consists of $(3 \times 6 = 18)$ blocks of size s in which s is a power of *two*. Each $s \times s$ block is an identity matrix that has been shifted to the right a_{mn} times, $m = 1, ..., 3, n = 1, ..., 6$. The shift values can be any value between 0 and $s - 1$, and have been determined with a heuristic search for the best performance in the codes of the same structure. Our approach is different from [31] since the sub-block length is not a prime number. Also, shifts are determined by simulations and searching for the best matrix that satisfies our constraints (with the highest girth [32]). Figure 5 shows a comparison between the performance of two sets of $(3, 6)$ LDPC codes of rate $1/2$ and block lengths of 768 and 1536 designed with the above structure. To give some comparison points, [10] uses a LDPC code of length 1020 which achieves BER of 10^{-5} and 0.3×10^{-5} for SNR of 2.7 and 3.1 dB, respectively.

3.1 Reconfigurable Architecture for Regular Codes

For LDPC codes, increasing the block length results in a performance improvement. This increases the error correction ability of the code. Having a reconfigurable

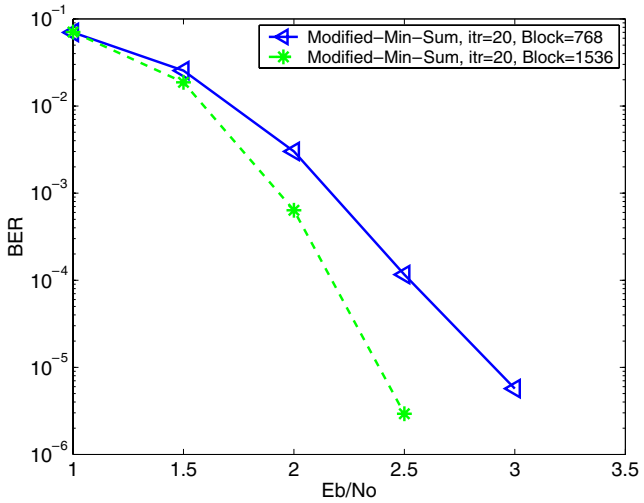

Figure 5 Simulation results for the decoding performance of different block lengths.

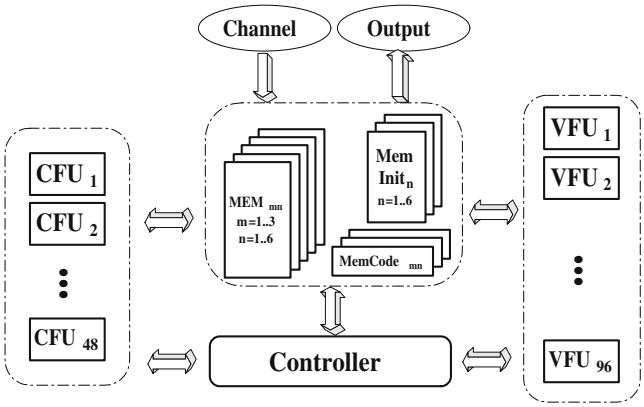

Figure 6 Overall architecture of a semi-parallel regular LDPC decoder.

architecture which can be scaled for different block lengths enables us to choose a suitable block length N for different applications. Usually N is in the order of $500 \sim 10000$ bits for practical uses. Our design is flexible for block lengths of $N = 6 \times 2^{\theta}$ for a (3,6) LDPC code. As an example for $\theta = 8$, N is equal to 1536. By choosing different values for θ we can get different values for the block length. We will discuss the statistics and design of the architecture for block length 1536 bits, as an example. The proposed LDPC decoder can be scaled for any block length of size $N = 6 \times 2^{\theta}$. The largest block length is determined with the physical limitations of the hardware platform such as FPGA or ASIC. It should be noted that in this architecture, changing the block length is an off-line process, since a new bitstream file should be compiled to be downloaded to the FPGA.

The block diagram of a regular $(3, 6)$ LDPC decoder is shown in Fig. 6. This semi-parallel architecture consists of $w_c \times w_r = 3 \times 6 = 18$ memory units $(MEM_{mn}, m = 1, ..., w_c, n = 1, ..., w_r)$ for storage of the check and variable messages and w_r memories $(MemInit_n)$ for storage of the initial values read from the channel. $MemCode_{mn}$ stores the decoded bits that result from each iteration of the decoding. This architecture has several processing units called VFUs and CFUs that are reused in each iteration. Since the code rate is $1/2$, there are twice as many columns in the PCM as rows, which means that the number of VFUs is two times the number of CFUs to balance the time spent on each half-iteration. For the block length of 1536, we have chosen the parallelism factor of $S = 16$, which means that there are $(1536 - 768)/16 = 48$ CFUs and 96 VFUs. Each of these units is used 16 times in each iteration. These units perform computations on different input sets that are synchronized by the controller unit.

Figure 7 shows the interconnection between memories, address generators and CFUs that are used in the first half of iterations. In each cycle $ADGC_{mn}$ generate addresses of the messages for the CFUs. Split/Merge (S/M) units pack/unpack messages to be stored/read to/from memories. To increase the parallelism factor, it is possible to pack more messages (i.e. δ) in a single memory location. This poses a constraint on the design of the H matrix, since the shift values should all be multiples of δ. The finite state machine 'control unit' supervises the flow of messages in/out of memories and functional units.

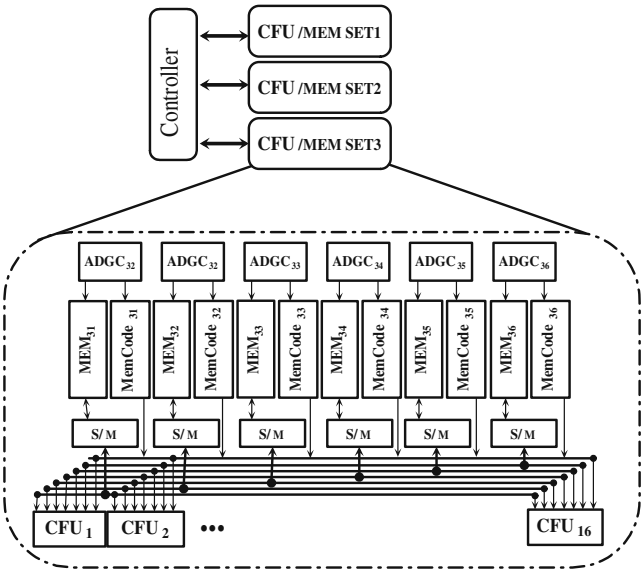

Figure 7 Connections between memories, CFUs and address generators.

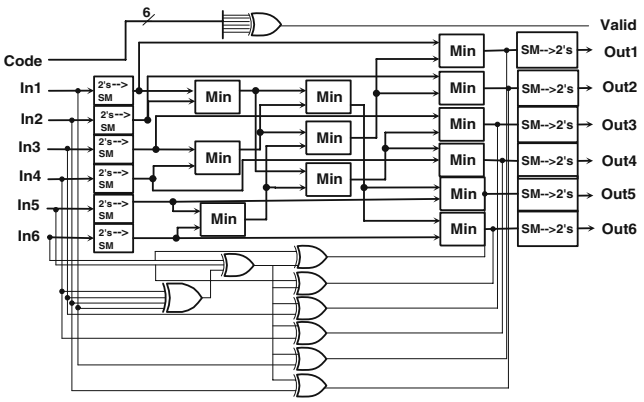

Figure 8 Parallel CFU architecture.

Figure 8 shows a parallel architecture for the CFUs. Each CFU has $w_r = 6$ parallel inputs and outputs. This unit computes the minimum among different choices of five out of six inputs. The CFU outputs the results to output ports corresponding to each input which is not included in the set. For example $out1$ is the result of:

$$out1 = \min(abs(in2), abs(in3), ..., abs(in6)). \quad (14)$$

in which $abs(.)$ is the absolute value function.

Also, during the computations of the current iteration, the CFU checks the code bits resulting from the previous iteration to check if they satisfy the corresponding parity check equation (step 5 of the decoding algorithm). After the first half of the iteration is complete, the result of all parity checks on the codeword will be ready. With this strategy, computations in check nodes and variable nodes can be done continuously without the need to wait for checking the codeword resulting from the previous iteration. This increases the speed of the decoding.

The interconnection between VFUs and memory units and address generator $ADGV$ is shown in Fig. 9.

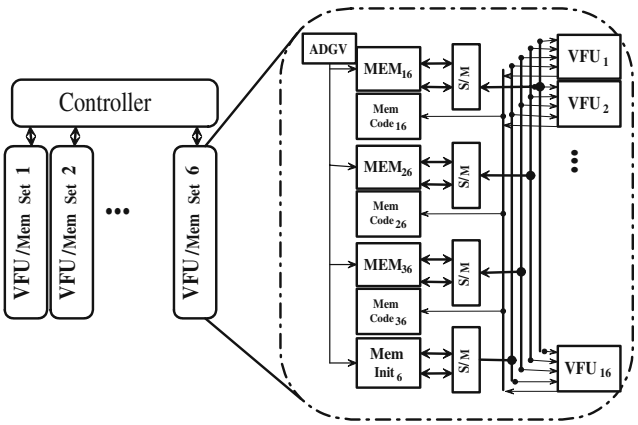

Figure 9 Connections between memories, VFUs and address generators.

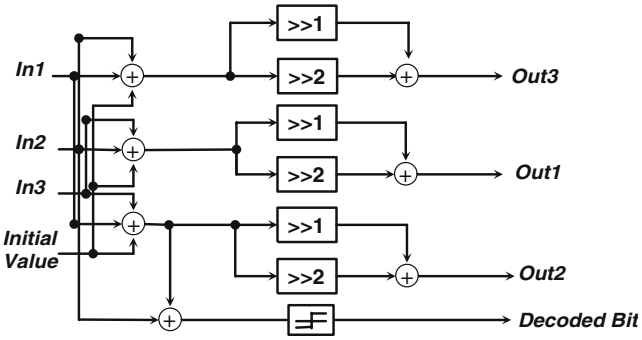

Figure 10 VFU architecture.

Locations of the messages in the memories are such that a single address generator can service all the VFUs. The controller makes sure that all the units are synchronized.

The architecture of a VFU is shown in the Fig. 10. This unit adds different combinations of its inputs and scales them with a scaling factor of 0.75 which is done with shift and addition. Also, it thresholds the summation of its inputs to find the code-bit corresponding to that variable node.

3.1.1 FPGA Architecture of Regular LDPC Codes

Most of the computations in a FPGA are fixed point instead of floating point because fixed point operators require smaller area. Fixed point operators suffer from quantization error. Increasing the number of bits in the operands decreases this error. There is a trade-off between the number of quantization bits, area of the design, power consumption and error-correcting performance. Using more bits decreases the bit error rate, but increases the area and power consumption of the design. Also, depending on the nature of the messages, the number of bits used for the integer or fractional part of the representation is important. Our simulations show that using 5 bits for the messages is enough for good performance. One bit is used for the sign and two bits for each of the integer and fractional parts.

Since the memory blocks in the FPGA have no more than two ports, we increase the number of the message read/writes in each clock cycle by packing eight message values and storing them in a single memory address. This enable us to read $2 \times 8 = 16$ messages per memory per cycle.

A prototype architecture has been implemented by writing Hardware Description Language code and targeted to a Xilinx Virtex4-xc4vfx60 FPGA. Table 2 shows the utilization statistics of the decoder on the FPGA. The maximum clock frequency of this decoder

Table 2 Design statistics for the (3,6) regular LDPC decoder on Virtex4-xc4vfx60 FPGA for a block length of 1536 bits and code rate 1/2.

Resource	Used	Utilization rate(%)
Slices	9,881	39
FFs	3,455	6
LUTs	18174	35
Block RAMs	66	28

is 211 MHz after place and route. Considering the parameters of our design, it takes 96 cycles to initialize the memories with the values read from the channel, 32 cycles for each of the CFU and VFU half-iterations, and 48 cycles to send out the resulting codeword. Based on the average number of decoding iterations required to achieve a frame error rate (FER) of 10^{-4}, the data rate of this decoder is 397 Mbps.

4 LDPC Architecture for Block-structured Irregular LDPC Codes

In this section, we describe the details of our second LDPC decoder. This decoder is more general than the first one and can be scaled to different code rates/ sizes on the fly. It is designed to support irregular block-structured PCMs proposed for the IEEE 802.11*n* wireless standard. Based on this standard, sub-block sizes are multiples of 27 and the code profiles are optimized to minimize the number of short cycles and therefore achieve excellent performance. As an example, a PCM of a code with a block length of 1296 bits and code-rate of 2/3 is shown in Fig. 11. The PCM is partitioned into square sub-matrices of size 54×54 with at most one nonzero entry per row/column. Each sub-matrix is either a shifted identity matrix or a zero matrix. The black squares show a shifted identity matrix and the white squares show a zero matrix. The PCMs in the standard are similar to the PCMs that we designed for the regular

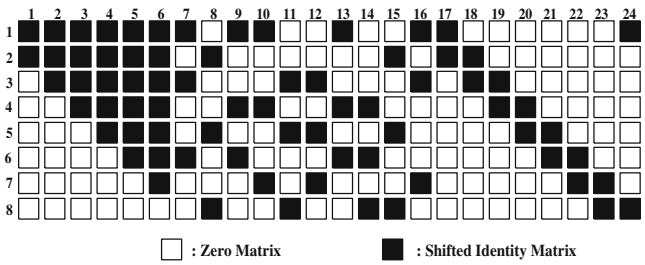

Figure 11 An overall view of a block-structured irregular parity-check matrix, $N = 1296$ bits and $R = 2/3$. The *black squares* show a shifted identity matrix and the *white squares* show a zero matrix.

LDPC decoder described in the previous section and Fig. 4. All of them support block-structured codes. The difference is that the PCMs in the standard support irregular codes and hence have some zero blocks which leads to more complex addressing in the decoder for irregular codes.

4.1 Scalable Decoder Architecture for Regular/Irregular Codes

In this section, we present the description of our configurable, flexible LDPC decoder implementation. Based on the standard and simulation results presented in Section 4.4, the following features are utilized: The LBP decoding algorithm is used since it converges twice as fast as the Sum-Product algorithm. This results in the same error- correcting performance with half the decoding iterations. The PCM is block-structured and irregular, which preserves the excellent performance of irregular random PCMs. It should be noted that designing an architecture for irregular codes is more challenging because it has to be flexible to support different column degrees. The decoder accepts block lengths of $N = 648, 1296, 1944$ bits as input, which corresponds to sub-block sizes of $S = 27, 54, 81$, respectively. It also supports four different code rates: $R = 1/2, 2/3, 3/4, 5/6$.

Figure 12 shows a block diagram of the LDPC decoder. The decoding is based on Eqs. 10, 11, and 12. The design consists of memory blocks (RAMs and ROMs) for storage of messages, processing blocks, permuters to route messages from memory to the processing units and a control unit similar to the design described in Section 3 and Fig. 6. Decoding starts by reading a parameter that selects the code rate and the code size of the input codeword block. According to this value, the appropriate values of the PCM are read from the Shift,

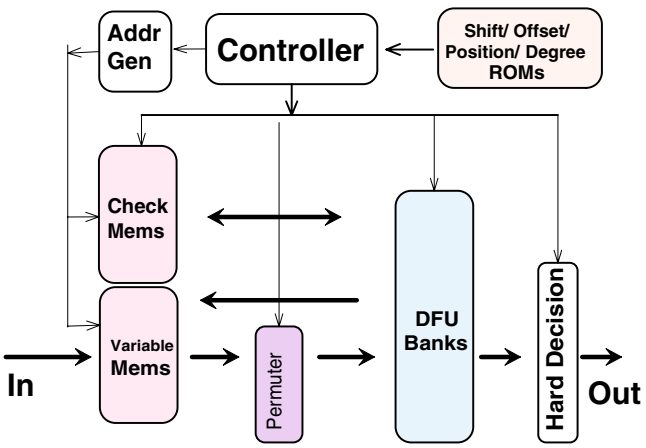

Figure 12 LDPC decoder block diagram.

Offset, Position and w_r ROMs. These values are used to generate the addresses to read/write from/in the check and variable memories. The decoder also inputs the reliability values and stores them in the variable memories. The variable values pass through the permuters to be routed to the correct decoding functional unit (DFU) for processing. In this architecture instead of VFUs and CFUs in Section 3, we have one processing unit that performs both computations. Hence, we call it the DFU. After the processing stage, results enter the 'early detection unit' (EDU) to check if the valid codeword is found. Also, the results are written back to the variable and check memories. The next layer/iteration starts when the decoder reads a new set of values from the variable memories. The EDU also analyzes the decoded word in parallel with the main processing path. The decoder stops and outputs the resulting codeword when either of the following two conditions is satisfied: either EDU detects a valid codeword, or the maximum number of iterations is reached. The control unit controls the flow of data between all the units. We will discuss each block of the architecture in more detail in the following sections.

4.1.1 DFU Banks

These blocks are the main processing part of the architecture since they update the check and variable messages and calculate new values. The number of parallel processing elements in these banks shows the parallelism factor of the design. For example, for the 1296 block code in Fig. 11, 54 processing nodes can work in parallel without any data dependency restrictions. To support different code lengths, the architecture should support a set of sub-block sizes of $S = 27, 54, 81$. Hence, we divide the DFUs into three banks, each of which contains 27 DFUs. For the size 1944 block, all three banks are used (81 DFUs), for the 1296 block, two of the banks are used (54 DFUs), and for the 648 block just one of the banks is in use. As mentioned before, this unit replaces both CFU and VFU processing units in the architecture in Section 3.

The DFU inputs w_r values from the check memories and w_r values from the variable memories, and updates new variable messages and check messages based on Eqs. 10, 11 and 12.

Figure 13 shows a block diagram of the DFU. The values of the variable and the check messages enter this unit and the difference is calculated based on Eq. 10. The sign and magnitude of inputs are separated and pass through separate computation paths based on Eq. 13. The serial Min-Sum unit inputs the absolute value of the variable messages and finds the

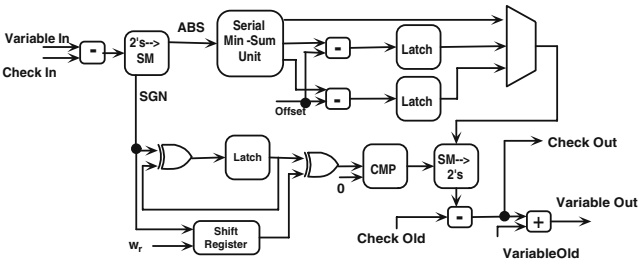

Figure 13 The DFU block diagram.

two minimums among the w_r values and the relative index of these values compared to the first input. Then, the offset value is subtracted from the minimums. The intermediate output corresponding to each input is the minimum of all the other values. It should be emphasized that this architecture uses the LBP with MMS with correcting offset, whereas, the regular LDPC architecture used the MMS with scaling factor. Both of these approaches result in the same improvement in the decoding performance and decrease in the computation requirements.

The sign of each output is the multiplication of the signs of all the inputs other than the input corresponding to this output. Then, the previous check node values are subtracted from these and the variable node values are added to the results to generate the final outputs (w_r of them) based on Eq. 12. The outputs of all the S-DFUs are concatenated and stored in one address of the variable memories. It is important to note that we do not permute back these values; variables are stored in the shifted order. The permuter in the next layer uses values from the Offset ROM instead of the original shift values. This way we eliminate the need for another permuter which saves 8% of the total area.

Serial Min-Sum Unit This unit inputs w_r values and finds the two smallest values. Depending on the requirements on the architecture, this unit can be designed in two different ways: 'Parallel–Input' (PI) or 'Serial–Input'. The PI generates results faster by using processing units in parallel (refer to Section 3 and Fig. 8). This approach is suitable for a decoder that supports regular LDPC codes with a fixed number of inputs. Since we are dealing with irregular codes, the number of nonzero sub-matrices in each row of the PCM is different for each layer. This means that the number of inputs for the Min-Sum unit varies based on the w_r. In the irregular PCMs for our architecture, these values are between 7 and 21 for different code rates. In order to support a variable number of inputs, we have designed a 'serial' Min-Sum unit. Although this unit runs slower than the parallel version, it has the flexibility of accepting any number of input values

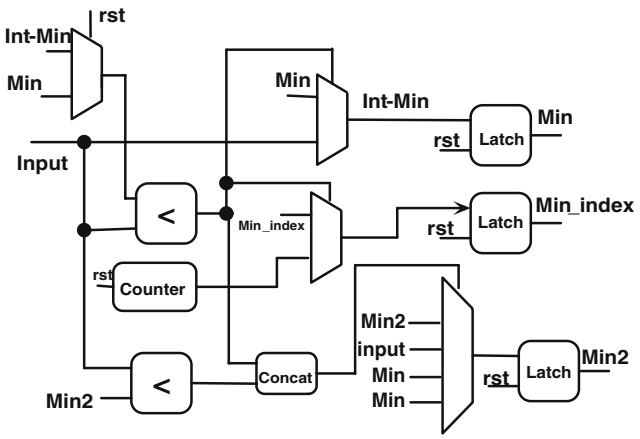

Figure 14 The serial Min-Sum unit block diagram.

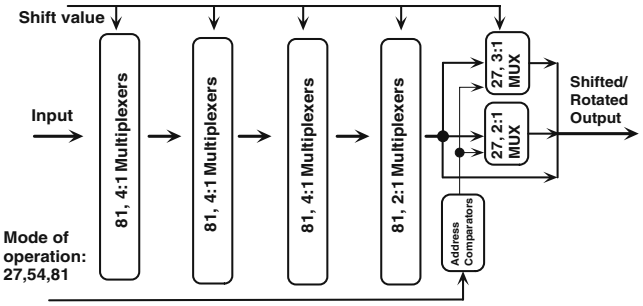

Figure 15 The flexible permuter block diagram.

in serial fashion using just one input port. In this way, the hardware can support irregular codes of different rates and sizes. The controller marks the beginning and end of the input sequence. The Min-Sum unit finds the two smallest numbers and their location in the sequence compared to the first input. Figure 14 shows the block diagram of this unit.

4.1.2 The Flexible Permuters

One of the main challenges of the LDPC decoder is 'routing' the messages from the memories to the correct processing units as quickly as possible. For a fixed decoder that supports a single block length/code rate,such as the decoder in Section 3, this is done by the address generators, split/merge units and routing wires. For a more general case, this can be done with a network of multiplexers to route the signals according to the shift values. We call these multiplexer networks a 'permuter'.

In this architecture, permuters are used to shift a block of S, b-bit numbers to generate the correct addressing based on the PCM. For example, a shift of s means that the order of the outputs should be $(s+1)\ldots S,1\ldots s$ instead of $1\ldots S$. Analysis was done on the structure of permuters to determine the best structure. We tested permuters of size 81 using different sizes/combinations of multiplexers, and selected the one with the smallest area and highest speed, which is designed with four levels of $4:1, 4:1, 4:1, 2:1$, multiplexers with b-bit inputs/outputs.

Since permuters occupy a significant portion of the area, instead of having three permuters to support different values of S (corresponding to different code rate/size), we designed a flexible permuter of size 81, which can be used to permute any of the sub-block sizes. This flexible permuter is made by adding a layer of multiplexers to the original permuter of size 81 and selecting the proper signal for each case as shown in Fig. 15.

4.1.3 Memory Organization

There are several ROM/ RAM blocks in the system. These blocks are divided into different banks for check and variable messages and a few ROMs that store the parameters to regenerate the various PCMs. To be able to read/write one full sub-block matrix per clock cycle, the check memory and the a posteriori memory need to be organized in the appropriate manner.

Three memory blocks are used to store the check messages. Organization of the check-node memory is shown in Fig. 16. In order to increase the throughput of the decoder and take advantage of the parallel processing, we use packed storage of the check and variable messages. S messages are concatenated and stored in a single memory address that can be read in a single clock cycle. Permuters are used to split and route each single message to the corresponding DFU unit. Also, we have divided these memory blocks into three modules that accept the same address (to avoid extremely large memory word lengths). Each of these modules packs and stores 27, b-bit values from each sub-block matrix per address. When $b = 8$ bits, each word in the memory will be $27 \times 8 = 216$ bits long. These values correspond to the check outputs for the DFUs numbered from 1 to 27, 28 to 54, and 55 to 81, respectively. In this way, by reading from a single address from three memory blocks, all the 81 check values are ready for concurrent processing.

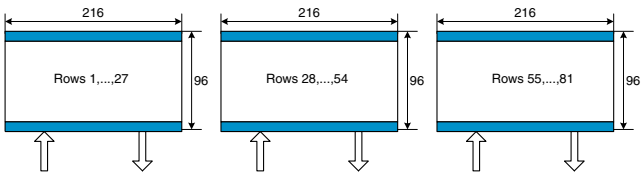

Figure 16 The check memory organization block diagram.

In the case of the largest codeword size of 1944 all three check memory sub-modules will be used, while only two or one module will be used in the cases of codeword sizes of 1296 or 648, respectively. The unused check memory modules can be turned-off. The depth of the check memory sub-modules depends on the number of layers and number of non-zero sub-block matrices per layer (row connectivity degree). The largest depth for code rate of 1/2 is 96 since there are (in average, because of the code irregularity) eight non-zero sub-block matrices per layer and there are 12 layers. The addressing of check messages is very simple since the memory locations are always accessed in an increasing order through the use of a hardware address increment unit.

Using the same packed storage concept, three dual port memories are used to store the variable messages. Organization of the memories that store the variable values is shown in Fig. 17. The variable values are also grouped and stored in a single memory address (groups of 27, b-bit numbers). During the initialization step, these messages are stored in the original order of the inputs. They are permuted based on the elements of the PCM to route to the correct DFU for processing. Depending on the block sizes, the variable memories can be divided into three or more modules.

We should note that because of the special structured design of the LDPC codes there is no memory access conflict either in the check memories or in the variable memories in the decoder architecture.

Packed Storage of Multiple PCMs Our flexible architecture supports $n = 3$ block lengths and $r = 4$ code rates which gives us $\eta = n \times r = 12$ combinations. These η PCMs that correspond to each code size/rate should be stored in ROM memories which can be very large if stored directly. The intuition behind reducing the memory size required for storing the PCMs is to take advantage of the structure of the PCMs and store a few parameters and regenerate the matrix when needed. Considering the structure of the PCM, we can rebuild it by knowing a few parameters: block length, sub-block size, number of nonzero sub-matrices in each layer of the PCM, position of the nonzero blocks and the shift value to generate the shifted identity matrices. The PCM can be regenerated on the fly using counters and permuters.

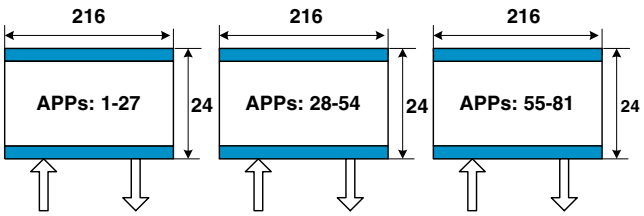

Figure 17 The variable memory organization block diagram.

We have added another value to these parameters namely 'offset' values. These values are the shift values with regard to the previous shift of the same layer (see Fig. 11). For example, if the shift value for layer k is s_k and the shift value for the next layer is s_{k+1}, the offset o_{k+1} will be equal to: $o_{k+1} = s_{k+1} - s_k$. The decoder uses the value of o_{k+1} in the permuter to route the messages.

4.1.4 Early Detection Unit

To detect that the correct codeword is found, two sets of tests are performed after finishing the decoding of each layer. First, check if all the parity check equations are satisfied ($H_l \times c = 0$). Then, compare the signs of the updated variable messages with the previous values of these messages and check if the signs have changed: $\left(sign(Lq_j^{k,l}).sign(Lq_j^{(k,l-p)})\right) > 0$.

The outputs of a layer are valid if all the parity check equations are satisfied and there is no sign change in the results. It is required that L (total number of layers) consecutive layers satisfy these two constraints even if they belong to two consecutive iterations. Then, decoding stops and the resulting codeword is output.

4.1.5 Controller

This block controls the flow of the messages into/out of different blocks during the decoding. The controller generates enable/reset and all the hand-shaking signals necessary for correct operation of the decoder. It also controls the counters that generate addresses for ROM and RAM memories. The controller inputs the values of block size and code rate and also reads the number of nonzero blocks in each layer (w_r) from the w_r ROM. Based on these parameters, this unit controls the flow of data to the DFUs, Min-Sum units, permuters and other blocks in the system. The main challenge of the controller is to keep track of the groups of w_r messages throughout the process and read/ process/ write them at the correct time.

4.2 Hardware Overhead

To support the highest data rates for wireless networks, this decoder is designed to support the largest block length ($N = 1944$). For the smaller block lengths some of the DFU units and memories are unused and can be turned off/disabled using clock gating. Because of the data dependency, it is not possible to use a number

Table 3 Design statistics for the flexible irregular LDPC decoder on Virtex4-xc4vfx60 FPGA.

Resource	Dec7	Dec8
Slices	11,328	12,633
FFs	12,368	13,823
LUTs	17,104	19,265
Block RAMs	87	87

of DFUs greater than the sub-block size. This is a limitation that is imposed by the PCM structure and the decoding algorithm. Another overhead in the design comes from storing twelve PCMs in the memories corresponding to each combination of the code rate/size, which is almost 22 Kbits. There is also 19% overhead for the flexible permuter compared to a fixed permuter of size 81.

4.3 Hardware Implementation of LDPC Decoder for Irregular Codes

Two prototype architectures for the LDPC decoder have been implemented in Xilinx System Generator and targeted to a Xilinx $Virtex4 - xc4vfx60$ FPGA. These cases include:

- LDPC decoder with 7-bit messages (Dec7)
- LDPC decoder with 8-bit messages (Dec8)

Table 3 shows the utilization statistics of these architectures. A clock frequency of 160 MHz is achieved for both of these designs after place and route.

The proposed decoder architecture is also synthesized for a Chartered Semiconductor 0.13µm, 1.2 V, CMOS technology using the BEE/Insecta design flow [33] and Synopsys tools. The Chartered memory compiler was used to generate efficient RAM and ROM blocks. Table 4 shows the area occupied by each part of the decoder with 8-bit messages in square millimeters. This architecture runs at a maximum clock speed of 412 MHz and consumes 502 mW of dynamic power (estimated using Design Compiler). The total area for the decoder is 2.2036 mm^2 for the Dec7 and 2.4928 mm^2 for the Dec8.

Table 4 ASIC design statistics for the Dec8 flexible LDPC decoder.

Resource	Area mm^2
Memory banks	1.1925
DFU banks	0.939
Permuters	0.289
Control	0.0029
LDPC decoder	2.4928

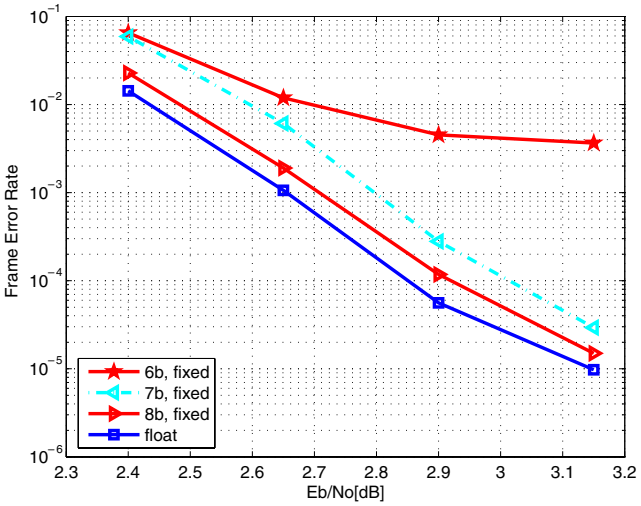

Figure 18 FER for irregular block-structured PCM (R=2/3, N=1296, MaxIter=15, LBP).

4.4 Decoding Throughput and Performance

Figure 18 shows the FER performance of the implemented decoder for a 2/3-rate code with a code length of 1296 bits and the LBP decoding algorithm. Performance results for 6, 7 and 8-bit fixed point arithmetic precisions and floating point are shown. Both 7 and 8-bit fixed-point precisions have small loss compared to the floating point version.

We have estimated the decoding throughput (considering the information bits) based on the average number of decoding iterations required to achieve a FER of 10^{-4}, while the maximum number of iterations is set to 15. The clock frequency is 160 MHz for the decoder running on the FPGA and 412 MHz for the ASIC implementation. Figure 19 shows the average throughput as a function of code rate and codeword size for the decoders. As an example, for a block length of 1944 and a rate 5/6 code, the decoder implemented

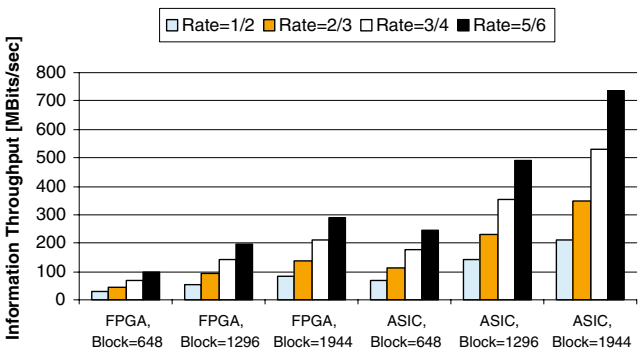

Figure 19 Average decoding throughput for different code rates and codeword lengths for the flexible irregular architectures synthesized for FPGA and ASIC based on the average number of iterations.

on the FPGA achieves an average throughput of 292 Mbits/sec and the ASIC version achieves 736 Mbits/s. Average FPGA latency of the decoder is between 5 and 11 μs for different block lengths/rates while the average ASIC latency is between 2.2 and 4.5 μs.

Now, we would like to compare the results of the two decoder architectures for the regular (refer to Section 3) and the flexible architecture for the irregular codes that we described in this section. The decoder for the regular (3, 6) LDPC code of rate 1/2 and block length 1536 bits achieves decoding throughput of 397 Mbps when running on the FPGA. The flexible decoder for irregular codes of rate 1/2 and block length of 1296 bits achieves a throughput of 56 Mbps, and the decoder for code rate 1/2 and block length of 1944 bits achieves a throughput of 84 Mbps. The regular decoder achieves much higher throughput than the flexible decoder for irregular codes at the expense of less flexibility.

5 Conclusions

We presented the design and hardware implementation of two decoder architectures for LDPC code. The first one supports (3, 6) regular codes and is implemented in parallel, having a higher throughput and smaller area. This architecture is suitable for applications in which throughput of the decoding is very important and scalability is not needed. The second decoder is a flexible architecture for structured irregular LDPC codes. This decoder supports a family of code sizes and rates and can switch between different cases on the fly. The decoder uses serial computation units which enhances the flexibility with the cost of increased latency. There is also some overhead in the memory and area usage of this decoder to support the 12 combinations of the code sizes and code rates. This decoder is suitable for the applications that require scalability. The decoder has a general structure and can support both regular and irregular codes. It can also be extended to support other code families. Both of the decoders are implemented on an FPGA and the flexible decoder is synthesized for ASIC.

Acknowledgements This work was supported in part by Nokia Corporation and by NSF under grants EIA-0321266, CCF-0541363, CNS-0551692 and CNS-0619767. We would like to thank Yang Sun for his help in ASIC synthesis. Also, we would like to thank the reviewers for their useful comments.

References

1. Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on Information Theory, 8*, 21–28, January.
2. MacKay, D., & Neal, R. (1996). Near Shannon limit performace of low density parity check codes. In *Electronic Letters, 32*, 1645–6, August.
3. Karkooti, M., & Cavallaro, J. R. (2004). Semi-parallel reconfigurable architectures for real-time LDPC decoding. In *IEEE international conference on information technology: Coding and computing, ITCC 2004*, April.
4. *IEEE 802.11 Wireless LANsWWiSE Proposal: High throughput extension to the 802.11 Standard*. IEEE 11-04-0886-00-000n.
5. Chung, S., Richardson, T., & Urbanke, R. (2001). Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation. *IEEE Transactions on Information Theory, 47*, 657–670, February.
6. Chung, S., Forney, G., Richardson, T., & Urbanke, R. (2001). On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters, 5*, 58–60, February.
7. Karkooti, M., Radosavljevic, P., & Cavallaro, J. R. (2006). Configurable high throughput irregular LDPC decoder architecture: Tradeoff analysis and implementation. In *IEEE 17th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 360–367), September.
8. Hocevar, D. E., (2004). A reduced complexity decoder architecture via layered decoding of LDPC codes. In *IEEE workshop on signal processing systems, SIPS* (pp. 107–112).
9. Blanksby, A., & Howland, C. (2002). A 690-mW 1-Gbps 1024-b, Rate-1/2 low-density parity-check code decoder. *Journal of Solid State Circuits, 37*, 404–412, March.
10. Zhang, T. (2002). *Efficient VLSI architectures for error-correcting coding*. PhD thesis, University of Minnesota, July.
11. Mansour, M. M., & Shanbhag, N. R. (2003). High-throughput LDPC decoders. *IEEE transactions on very large scale integration (VLSI) Systems, 11*, 976–996, December.
12. Darabiha, A., Carusone, A. C., & Kschischang, F. R. (2005). Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity. In *IEEE international symposium on circuits and systems, ISCAS 2005*, May.
13. Kienle, F., Brack, T., & Wehn, N. (2005). A synthesizable IP core for DVB-S2 LDPC code decoding. In *Proceedings of design, automation and test in Europe*.
14. Fanucci, L., Rovini, M., L'Insalata, N. E., & Rossi, F. (2005). High-throughput multi-rate decoding of structured low-density parity-check codes. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E88-A*, 3539–3547, December.
15. Yang, L., Lui, H., & Shi, C.-J. R. (2006). Code construction and FPGA implementation of a low-error-floor multi-rate low-density parity-check code decoder. *IEEE Transactions on Circuits and Systems I, Regular Papers, 53*, 892–904.
16. Gunnam, K., Weihuang, W., Kim, E., Choi, G., & Yeary, M. (2006). Decoding of array ldpc codes using on-the-fly computation. In *IEEE asilomar conference on signals, systems and computers*, November.
17. Sun, Y., Karkooti, M., & Cavallaro, J. R. (2007). VLSI decoder architecture for high throughput, variable block-size

and multi-rate LDPC codes. In *IEEE international symposium on circuits and systems (ISCAS)*, May.
18. Wang, Z., & Cui, Z. (2007). Low-complexity high-speed decoder design for quasi-cyclic LDPC codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15*, 104–114.
19. Masera, G., Quaglio, F., & Vacca, F. (2007). Implementation of a flexible LDPC decoder. *IEEE Transactions on Circuits and Systems II: Express Briefs, 54*, 542–546.
20. Oh, D., & Parhi, K. (2007). Efficient highly-parallel decoder architecture for quasi-cyclic low-density parity-check codes. In *IEEE international symposium on circuits and systems (ISCAS)*, May.
21. Cui, Z., & Wang, Z. (2007). Efficient message passing architecture for high throughput LDPC decoder. In *IEEE international symposium on circuits and systems (ISCAS)*, May.
22. Sha, J., Gao, M., Zhang, Z., li, L., & Wang, Z. (2006). Efficient decoder implementation for QC-LDPC codes. In *International conference on communications, circuits and systems* (Vol. 4), June.
23. Shimizu, K., Ishikawa, T., Togawa, N., Ikenaga, T., & Goto, S. (2006). A parallel LSI architecture for LDPC decoder improving message-passing schedule. In *IEEE international symposium on circuits and systems (ISCAS)*, May.
24. Zhu, Y., Chen, Y., Hocevar, D., & Goel, M. (2006). A reduced-complexity, scalable implementation of low density parity check (LDPC) decoder. In *IEEE workshop on systems design and implementation and signal processing (SIPS)* (pp. 83–880), October.
25. Zhu, Y., & Chakrabarti, C. (2006). Aggregated circulant matrix based LDPC codes. In *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, May.
26. Oh, D., & Parhi, K. (2006). Low complexity implementations of sum-product algorithm for decoding low-density parity-check codes. In *IEEE workshop on signal processing systems design and implementation (SIPS)* (pp. 262–267), October.
27. Richardson, T., & Urbanke, R. (2001). Efficient encoding of low-density parity check codes. *IEEE Transactions on Information Theory, 47*, 638–656, February.
28. Heo, J. (2003). Analysis of scaling soft information on low density parity check codes. *Electronics Letters, 39*, 219–221, January.
29. Chen, J., Dholakai, A., Eleftheriou, E., Fossorier, M., & Hu, X. (2005). Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications, 53*, 1288–1299, August.
30. Radosavljevic, P., de Baynast, A., & Cavallaro, J. R. (2005). Optimized message passing schedules for LDPC decoding. In *IEEE 39th asilomar conference on signals, systems and computers* (pp. 591–595), November.
31. Mansour, M., & Shanbhag, N. (2002). Low power VLSI decoder architectures for LDPC codes. In *Proc. of the int. symp. on low power electronics and design* (pp. 284–289).
32. Mao, Y., & Banihashemi, A. (2001). A heuristic search for good low-density parity-check codes at short block lengths. In *IEEE Int. Conf. on Comm.* (pp. 41–44), June.
33. Chang, C., Kuusilinna, K., Richards, B., Chen, A., Chan, N., Brodersen, R. W., & Nikoliae, B. (2003). Rapid design and analysis of communication systems using the BEE hardware emulation environment. In *14th IEEE international workshop on rapid systems prototyping*, June.

Marjan Karkooti received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1997, the M.S. degree in socioeconomic systems engineering from the Institute for Research in Planning and Development, Tehran, Iran, in 2000, the M.S. degree in computer engineering from Rice University, Houston, Texas in 2004. She is currently a research assistant in Center for Multimedia Communications at Rice University and is working towards her PhD degree in computer engineering. Her research interests include wireless communications, error correcting codes such as Low Density Parity Check codes and Turbo codes, cooperative communications, hardware design using FPGAs/ASIC.

Predrag Radosavljevic was born in Belgrade, Yugoslavia, in 1975. He received the M.Sc. degree in 2004 and the Ph.D. degree in 2008, from the Department of Electrical and Computer Engineering, Rice University, Houston, USA. His research interest include the design of detection and decoding algorithms and architectures for wireless communication systems. He is currently a technical advisor with Patterson&Sheridan, LLP.

Joseph R. Cavallaro received the B.S. degree from the University of Pennsylvania, Philadelphia, Pa, in 1981, the M.S. degree from Princeton University, Princeton, NJ, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, in 1988, all in electrical engineering. From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, NJ. In 1988, he joined the faculty of Rice University, Houston, Tex, where he is currently a Professor of electrical and computer engineering. His research interests include computer arithmetic, VLSI design and microlithography, and DSP and VLSI architectures for applications in wireless communications. During the 1996–1997 academic year, he served at the USA National Science Foundation as Director of the Prototyping Tools and Methodology Program. During 2005, he was a Nokia Foundation Fellow and a Visiting Professor at the University of Oulu, Finland. He is currently the Associate Director of the Center for Multimedia Communication at Rice University. He is a Senior Member of the IEEE. He was Cochair of the 2004 Signal Processing for Communications Symposium at the IEEE Global Communications Conference and General Cochair of the 2004 IEEE 15th International Conference on Application-Specific Systems, Architectures and Processors (ASAP).